E-Infrastructures
H2020-EINFRA-2015-1

EINFRA-5-2015: Centres of Excellence
for computing applications

EoCoE

Energy oriented Center of Excellence
for computing applications

Grant Agreement Number: EINFRA-676629

D1.7 - M28
Software Technology Improvement

Project and Deliverable Information Sheet

EoCoE	Project Ref:	EINFRA-676629
	Project Title:	Energy oriented Centre of Excellence
	Project Web Site:	http://www.eocoe.eu
	Deliverable ID:	D1.7 - M28
	Lead Beneficiary:	Juelich JSC
	Contact:	Paul Gibbon
	Contact e-mail:	p.gibbon@fz-juelich.de
	Deliverable Nature:	Report
	Dissemination Level:	PU*
	Contractual Date of Delivery:	M28 01/31/2018
	Actual Date of Delivery:	06/29/2018
	EC Project Officer:	Carlos Morais-Pires

* - The dissemination level are indicated as follows: PU – Public, CO – Confidential, only for members of the consortium (including the Commission Services) CL – Classified, as referred to in Commission Decision 2991/844/EC.

Document Control Sheet

Document	Title:	Software Technology Improvement
	ID:	D1.7 - M28
	Available at:	http://www.eocoe.eu
	Software tool:	LaTeX
Authorship	Written by:	Matthieu Haefele (MdlS), Luc Giraud (INRIA), Leonardo Bautista Gomez (BSC), O. Abramkina (CEA/MDLS), Daniel Ruiz (IRIT), Yvan Notay (ULB), Pasqua D'Ambra (CNR), G. Marait (Inria)
	Contributors:	Kai Keller (BSC), Maciej Brzeźniak (PSNC), Karol Sierociński (PSNC), Tomasz Paluszkiewicz (PSNC), Julien Bigot (CEA), R. Lacroix (CNRS/IDRIS), Y. Meurdesoif (CEA/LSCE), M.H. Nguyen (CNRS/LSCE), Iain Duff (RAL-CERFACS), Philippe Leleux (CERFACS), Fahreddin Sukru Torun (IRIT-CNRS), Daniela di Serafino (UNICampania), Salvatore Filippone (Cranfield University), Ambra Abdullahi Hassan (UNITOV), E. Agullo (Inria), L. Giraud (Inria), M. Kuhn (Inria), L. Poirel (Inria)
	Reviewed by:	Paul Gibbon (JSC)

Contents

1. Document release note

This document is the first report on software technology improvement. Some activities are already implemented and some others are still on going work. The final document D1.8 due for M36 will replace this document and contain the final status of all activities that have taken place in EoCoE.

2. Motivation

From the outset, the EoCoE project was equipped with a diverse set of HPC expertise in WP1 designed to tackle a variety of possible performance bottlenecks in the applications from the four domain pillars. These range from state-of-the-art computer science tools for performance analysis, parallel IO etc..., to advanced linear algebra and other applied mathematics methods. This permits a layered approach to application tuning, starting from initial blind analysis to identify problematic code portions, then subsequently delving deeper to undertake complete refactoring of critical, compute-intensive routines. The key feature of EoCoE has been the close interaction between WP1 and the application domains WP2-WP5, enabling real-world energy applications to effectively exploit the existing European computing infrastructure and better equip them for future hardware advances. Ultimately we expect this work to expedite advances in simulations of low-carbon energy systems and technology.

This deliverable gathers the status of software technology advances conducted within the project. By software technology we mean specific computer science libraries or packages used in the scientific applications developed by EoCoE partners. The packages supported in EoCoE - such as the linear algebra libraries AGMG and PSBLAS - existed before the project and will continue to exist after it formally ends. Typically this software has been developed as part of a research project and as such, is not always mature enough in term of software engineering and robustness. The aim of the activities conducted here is to improve this situation and bring these packages closer to production-readiness.

3. Fault Tolerance Interface

3.1 Package ID card

Package name	Fault Tolerance Interface (FTI)
Functionalities offered	Multilevel Checkpointing in multiple formats and
Description	FTI is a multilevel checkpointing library with multiple features to reduce the stress on the parallel file system and reduce checkpointing overhead.
Number of users	1-10
Library dependencies	CMake, MPI
Package references	https://github.com/leobago/fti
Contact	• Leonardo Bautista Gomez (leonardo.bautista@bsc.es) • Kai Keller (kai.keller@bsc.es)

FTI stands for Fault Tolerance Interface[1] and is a library that aims to give computational scientists the means to perform fast and efficient multilevel checkpointing in large scale supercomputers. FTI leverages local storage plus data replication and erasure codes to provide several levels of reliability and performance. FTI is application-level checkpointing and allows users to select which datasets needs to be protected, in order to improve efficiency and avoid wasting space, time and energy. In addition, it offers a direct data interface so that users do not need to deal with files and/or directory names. All metadata is managed by FTI in a transparent fashion for the user. If desired, users can dedicate one process per node to overlap fault tolerance workload and scientific computation, so that post-checkpoint tasks are executed asynchronously.

3.2 Improvement achieved

Contributors	Leonardo Bautista Gomez (BSC), Kai Keller (BSC), Maciej Brzeźniak (PSNC), Karol Sierociński (PSNC), Tomasz Paluszkiewicz (PSNC), Julien Bigot (CEA)

During the reporting period several improvements were proposed to the FTI library implementation based on the automated code analysis, manual code review and testing the library with the built-in tests, dedicated testing applications as well as by integrating FTI with the Gysela application. The following paragraphs provide the details of this work.

First of all, the FTI library has been integrated with the continuous integration and static code analysis tools including Travis CI and Coverity scan. This enabled a more systematic approach to the further library improvement work.

In the second stage an extensive code analysis was conducted. It started with a static analysis of the library using Coverity scan and cppchek. At this stage 80 problems were found, mainly related to memory management, such as failure to free allocated memory segments of other resources. These problems were fixed and merged and to the code base. Another angle of the code analysis was to investigate MPI calls using a MUST checker. Within this analysis 2 problems were found and solved. Next the library I/O behaviour was checked using Darshan. As a result it was suggested to change the way of writing the level 4 checkpoints, by avoiding creating checkpoint file for each of the running processes, as this led to excess number of the checkpoint files.

In the third phase, the library built-in examples were used for testing the library. While running these tests, several run-time problems were found and fixed, including crashes during the recovery process or invalid handling of the input options as well as failure to take checkpoints in several situations. Most of these issues were fixed and fixes were merged to the main branch of the code.

In the fourth phase, a code refactoring was. It included splitting the source code into the functionally independent subprojects as well as unifying the code building approach.

Above mentioned activities were performed by PSNC with the aid and in consultancy with the FTI library developers. FTI-Gysela integration. Another work related to FTI library was to integrate it with Gysela, a scalable computing application. This work was performed by CEA and FZJ. At PSNC several tests were performed based on the benchmarks integrated with the application, leading to several improvements of the library.

First of all, comparison of the execution time of several Gysela workflows with and without FTI-based checkpointing was conducted. Within these tests both synchronous and asynchronous mode of the library operation were tested (note that in the async mode the dedicated processes, i.e. one per node, are created, and they are used for taking asynchronous checkpoints). Weak scaling was also tested. The tests were conducted in two phases: smaller test cases (up to 128 nodes of the Eagle cluster at PSNC) as well as bigger test cases (256 and more nodes). In the small-scale tests no major differences of the execution time (with and without FTI) were observed. These tests however had to be repeated because of using improper input values for Gysela during the first approach to testing. There was also an attempt to run bigger scale tests, however (as of Nov 2016) most of them failed due to the several repeating problems with the Eagle cluster (related to the infrastructure issues, external to the project activities). These test might need to be repeated in future.

In the most recent phase of the FTI library testing six testing applications were developed and run along with the FTI library with different configuration files for the library (the file determines e.g. the mode of taking checkpoints: synchronous vs asynchronous etc.). In the following paragraphs details of the testing applications are provided.

The first test (addInArray) uses basic FTI functions in order to make checkpoints and restarts the program from the last saved checkpoint. The aim of this test is to check if recovery is successful and all protected variables are correctly recovered (note: protected variables are those that are 'marked' to be included in the checkpoints). The recovered values are compared with the values expected at a given iteration (acquired by a full, non-interrupted execution of the testing application). Within the second test (diffSizes) every of the running processes (X-Y) expands its array (Realloc), and notifies the FTI about resizing the variable (by using relevant FTI function) and changes values written in it. Even ranks have 3 times larger array than odd ranks. After several iterations the program is stopped and restarted from the last checkpoint written. After the restart, it is checked if recovery is successful. At the end all the processes send their arrays to root process that checks if the results are correct. Problems with recovery after variable size change were notified while performing the tests. The new FTI function (FTI_Realloc) was proposed in order to solve this issue, by enabling to explicitly notify the FTI library about the fact the size of the variable was changed. Within the third test (heatdis) the examples from the FTI library are used in order to check correctness of the operation of the FTI functions such as FTI_Snapshot. Similarly to other tests the application is restarted after the simulated failure using the last checkpoint as an input. In this particular test, we tested if the snapshot functionality works properly despite scaling up the job size. It is important to note the snapshot function is designed to make a decision whether the actual checkpoint operation should be performed/triggered or not. In the current design and implementation the decision is based on the time criteria. During our tests we proved that triggering the checkpoints should not be based on the time criteria only as it is general it is hard to predict the application total execution time (or the time needed for particular iterations) if the job size is scaled up. Within the fourth test (lvlsRecovery) we examine the application recovery using all the checkpoint levels defined in the library. The computing job is stopped after some iterations instead of using FTI_Finalize function. In that way we keep all the levels saved on the persistent storage, while using FTI_Finalize function would cause removing the checkpoints made on different levels (L1, L2, L3). This lets us testing recovery from these various levels. The fifth test (nodeFlag) makes all the levels of the checkpoint and searches across the log files in order to make sure that there is only one process per node that goes through FTIs nodeFlag condition section. Example situation where such approach is needed is changing the folder for the storing the checkpoints. In that case only one process can make the change (this constitutes a 'critical section'). The sixth test (tokenRing) is very similar to addInArray test, but it uses FTI option to protect structures instead single variables. Within the tests some synchronization issues in the FTI library were discovered (e.g. some processes tried to use files or folder that did not exist yet) and fixes are provided.

In the last period the efforts on improving the FTI testing scripts and procedures were continued. In particular scripts for testing the FTI were created and add new tests were added to the automated testing mechanisms. The Travis CI configuration files were expanded in order to make automatic tests after every commit. We also managed to use 3 different compilers: gcc, clang and icc for compiling the FTI library and the testing applications, which improved scope and directions of the FTI testing.

References

[1] Leonardo Arturo Bautista-Gomez and al. Fti: High performance fault tolerance interface for hybrid systems. In *International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, 2011.

Acknowledgement

In case of PSNC the scientific/academic work is co-financed from financial resources for science in the years 2016-2018 granted for the realization of the international project financed by Polish Ministry of Science and Higher Education (agreement number 3543/ H2020/2016/2)

4. XML IO Server (XIOS)

4.1 Package ID card

Package name	XIOS
Functionalities offered	IO server and online post-processing
Description	XIOS is a hierarchical data managment library created by CEA/LSCE to handle its large needs in terms of data flow. Its asynchronous and flexible implementations enhance especially the uncoupling between computing needs ands data managment.
Number of users	The french climate community and a growing part of the european one.
Library dependencies	MPI, HDF5, NetCDF4
Package references	`http://forge.ipsl.jussieu.fr/ioserver`
Contact	• O. Abramkina (olga.abramkina@cea.fr) • R. Lacroix (remi.lacroix@idris.fr) • Y. Meuredesoif (yann.meurdesoif@cea.fr)

4.2 Improvement achieved

Contributors	O. Abramkina (CEA/MDLS), R. Lacroix (CNRS/IDRIS), Y. Meurdesoif (CEA/LSCE), M.H. Nguyen (CNRS/LSCE)

To be able to provide XIOS to a larger spectrum of applications than climate simulations, it was necessary to release some contraints on the XIOS implementation. Some on the heart of the library, like for the managment of grids or calendars, some on the output file backend. UGRID will illustrate this last point.

4.3 Grids composition

While a grid in previous versions of XIOS could only be composed of maximum one domain (a 2D plan, structured or unstructured) and one axis, XIOS is not any more limited to 3-dimension grids.

By allowing a grid to contain many domains and axis, XIOS provides a simple way to create high dimension grids. Moreover, with a new syntax, defining a multidimensional grid is easier than ever. For example, definition of a 6-dimension grid, as GYSELA's, can be done as following :

```
<grid>
<axis id="axis1"/>
<axis id="axis2"/>
<axis id="axis3"/>
<axis id="axis4"/>
<axis id="axis5"/>
<axis id="axis6"/>
</grid>
```

Users can easily define their own distribution of a grid by specifying the distribution of composing domain and/or axis. This deep modification has been the opportunity to also allow "zero-dimension" grids or scalar, which makes XIOS a tool to process and write various range of data.

Concerning grids, another obligatory “climate-specific” specification is lightened. In this way some meta-data related to longitudes and latitudes are optional, users choose the way to write out their data and associated meta-data.

4.4 Timeline managment

Since XIOS was originally developed to help dealing with the huge mass of data produced by climate simulations, the way it handled the simulation date and time was quite application-specific. Climate simulations are often used to study the evolution of the climate on Earth for large time scale, ranging from a few years to hundreds of years, with daily, monthly and/or yearly output frequencies. Due to this context, XIOS provided only Earth-based calendars and managed dates (for example the start date of the simulation) only as a fully-specified Earth date and time with the following format: “yyyy-mm-dd hh:mm:ss”.

Although this calendar system was well-suited for climate simulations, it did not make much sense for some other simulations, for example those with a small simulation time or non Earth-based. In order to open XIOS to other scientific communities, we modified the calendar system so that is more flexible.

Some elements that used to be mandatory like the start date of the simulation are now optional to ease the configuration of simulations that are not tied to a specific date. In addition, the date/time format was reworked to allow partial date/time definition, for example with just a year or a date. It also allows defining an optional offset expressed as a duration (for example “2015-01-11 12:00:00 + 1d” or “2017 + 42h11m”). Being that the date/time definition can be completely omitted, it is possible to only specify the duration offset, making XIOS virtually calendar-free.

Additionally, we added a fully customizable calendar (possibly month-free and with leap-year support) that can be configured to be suitable for planets other than the Earth.

4.5 Unstructured extension : UGRID

A new file output format has been implemented into XIOS to meet the needs of communities working with unstructured grids. It follows the UGRID conventions for netCDF file format [1] and it allows users to store the topology of the underlying unstructured mesh. Currently XIOS supports 2D unstructured meshes of any shape (triangular, quadrilateral, etc) and their mixture.

A 2D mesh can be described in the simpest case by a set of points, or nodes in the UGRID terminology, and/or by a set of edges and faces. XIOS allows one to define data on any of these three types of elements (nodes, edges, and faces). XIOS generates a full list of connectivity attributes proposed by the UGRID conventions. For example, in case of a mesh composed of faces the stored connectivity attributes will be the following:

This work has been integrated into the LFRic model developed by the UK Met Office. Preliminary tests of the LFRic with XIOS on the I/O end on the Met Office Cray super-

```
edge_node_connectivity
face_node_connectivity
edge_nodes_connectivity
face_nodes_connectivity
face_edges_connectivity
edge_face_connectivity
face_face_connectivity
```

computer reveal good I/O performances. These results will be presented at ParCo2017, an international conference on HPC.

References

[1] Ugrid conventions (v1.0). `http://ugrid-conventions.github.io/ugrid-conventions/`.

5. ABCD

5.1 Package ID card

Package name	ABCD
Functionalities offered	Parallel sparse hybrid iterative and direct solver
Description	ABCD (Augmented Block Cimmino Distributed Solver) is a distributed hybrid (iterative/direct) solver for sparse linear systems.
Number of users	1-10
Library dependencies	MPI, MUMPS, BLAS, LAPACK, PaToH, Boost
Package references	https://bitbucket.org/apo_irit/abcd
Contact	• Daniel Ruiz (daniel.ruiz@enseeiht.fr) • Fahreddin Sukru Torun (ftorun@enseeiht.fr) • Philippe Leleux (leleux@cerfacs.fr)

ABCD Solver consists of two parallel methods which are parallel hybrid block Cimmino iterative method and parallel augmented block Cimmino which is a pseudo-direct method. Both methods solve sparse systems of linear equations of the form $Ax = b$, where A is a square sparse matrix, on distributed memory computers.

Parallel Block Cimmino Hybrid Iterative Method

This method follows the well-known block Cimmino method: a row projection method for solving linear systems, see [2] for more details. In this method $Ax = b$ is partitioned as blocks of rows:

$$\begin{pmatrix} A_1 \\ A_2 \\ \vdots \\ A_p \end{pmatrix} x = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \end{pmatrix}. \tag{1}$$

and then the algorithm computes a solution iteratively from an initial estimate $x^{(0)}$ according to:

$$x^{(k+1)} = x^{(k)} + \omega \sum_{i=1}^{p} A_i^+ \left(b_i - A_i x^{(k)} \right). \tag{2}$$

Figure 1 shows a geometrical point of view of a sample iteration of Cimmino algorithm when there is two partitions.

The iterations can be reformulated as:

$$\begin{aligned} x^{(k+1)} &= \left(I - \omega \sum_{i=1}^{p} A_i^+ A_i \right) x^{(k)} + \omega \sum_{i=1}^{p} A_i^+ b_i \\ &= Q x^{(k)} + \xi, \end{aligned} \tag{3}$$

where $\xi = \omega \sum_{i=1}^{p} A_i^+ b_i$ and $Q = I - \omega \sum_{i=1}^{p} A_i^+ A_i$. Looking at the stationary point, this is equivalent to the linear system

$$Hx = \xi, \tag{4}$$

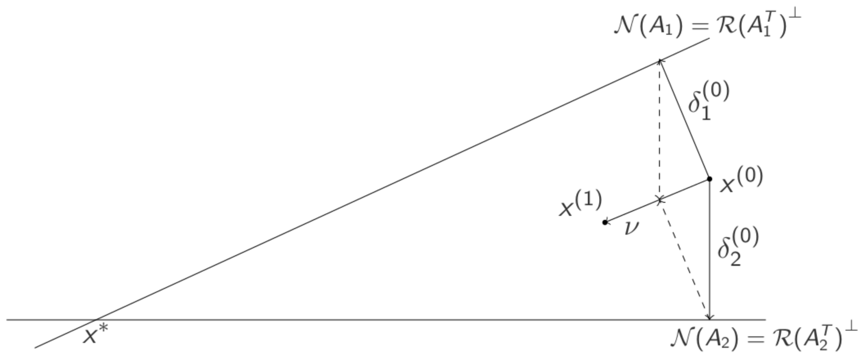
$\mathcal{N}(A_1) = \mathcal{R}(A_1^T)^\perp$
$\delta_1^{(0)}$
$x^{(0)}$
$x^{(1)}$
ν
$\delta_2^{(0)}$
x^*
$\mathcal{N}(A_2) = \mathcal{R}(A_2^T)^\perp$

Figure 1: Geometric point of view of the block Cimmino Algorithm with $p = 2$

where $H = I - Q$. Since H is symmetric positive definite we can solve this system by using Conjugate Gradient (CG) iterative method. The CG accelerated block Cimmino algorithm is studied in details [2, 3, 4, 7]. One of the issues in the CG iteration is to compute the projections onto A_i^T. The chosen method is through the solution of an augmented system [1] of the form

$$\begin{pmatrix} I & A_i^T \\ A_i & 0 \end{pmatrix} \begin{pmatrix} u_i \\ v_i \end{pmatrix} = \begin{pmatrix} 0 \\ r_i \end{pmatrix}, \tag{5}$$

where $r_i = b_i - A_i x^{(k)}$. The solution subvector u_i of the augmented system gives the projection. These systems are symmetric indefinite and we can solve them using the direct parallel solver MUMPS, which makes efficient use of the parallelism and gives to our solver the hybrid property. Our goal in this solver is then to have partitions capturing the ill-conditioning of the matrix that will be tackled by the direct solver so that the CG can converge quickly.

Figure 2 illustrates the execution steps of the parallel block Cimmino algorithm. In the algorithm, if there are more MPI processes than row-blocks, ABCD adopts a master-slave approach for the distributed solution of the system. Each master processor owns one row-block and creates an augmented system which is assigned to one MUMPS instance, referred as master. Then each slave processor is assigned to a master processor with respect to load criteria. More slaves are assigned to highly loaded master processors. The slave processes are exploited to cooperate with the masters' factorization and solution within MUMPS.

The convergence of the block Cimmino iterative method depends heavily on the angles between the subspaces determined by the row-block partitioning. Intelligent row-block partitioning methods are proposed [5, 8] in order to improve the convergence of block Cimmino method. In the extreme case where subspaces would be orthogonal, only one iteration would be necessary to get to the solution [6] (pseudo-direct solver). In the next subsection, we will elaborate this method.

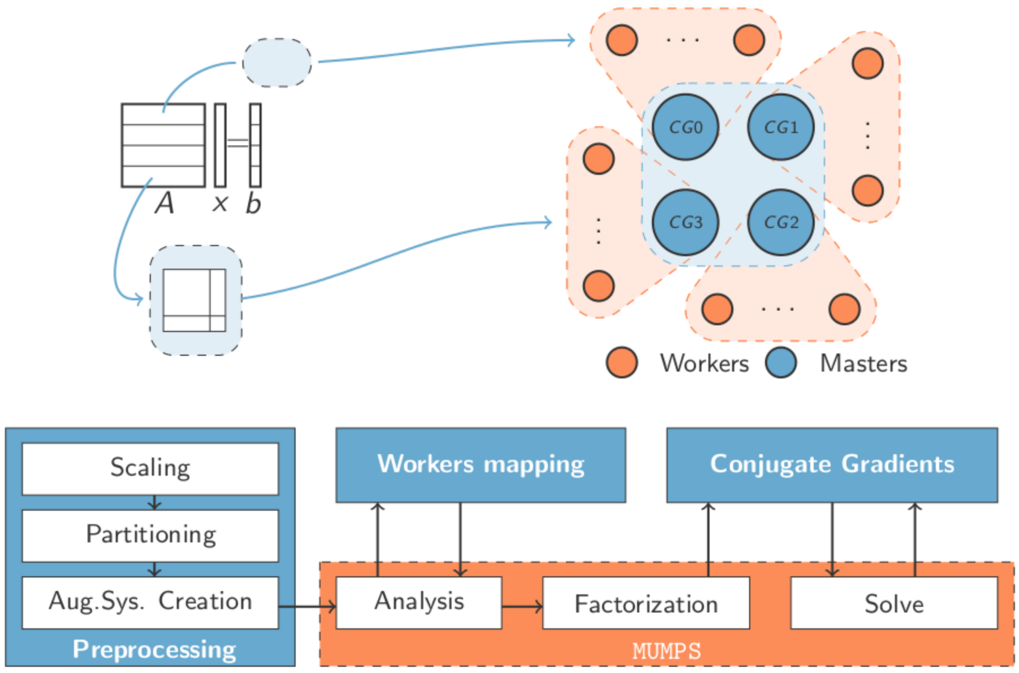

Figure 2: Execution steps of the parallel block Cimmino distributed solver

Parallel Augmented Block Cimmino Pseudo-direct Method

To understand the augmented block Cimmino algorithm, suppose that we have a matrix A with three partitions, described as follows:

$$\begin{bmatrix} A_{1,1} & A_{1,2} & & & A_{1,3} \\ & A_{2,1} & A_{2,2} & A_{2,3} & \\ & & A_{3,2} & A_{3,3} & A_{3,1} \end{bmatrix}, \tag{6}$$

where $A_{i,j}$ the sub-matrices of A_i, i-th row-block partition, that is interconnected algebraically to the partition A_j, and vice versa.

The goal of the augmented block Cimmino algorithm is to make these three partitions mutually orthogonal to each other, meaning that the inner product of each pair of partitions is zero. We consider two different ways to augment the matrix to obtain these zero matrix inner products.

The first way to augment the matrix to make all the partitions mutually orthogonal to each other is obtained by putting the product $C_{ij} = A_{ij}A_{ji}^T$ on the right of the partition A_i and adding $-I$ on the right of A_j viz.

$$\bar{A} = \left[\begin{array}{ccccc|ccc} A_{1,1} & A_{1,2} & & & A_{1,3} & C_{1,2} & C_{1,3} & \\ & A_{2,1} & A_{2,2} & A_{2,3} & & -I & & C_{2,3} \\ & & A_{3,2} & A_{3,3} & A_{3,1} & & -I & -I \end{array}\right]$$

The second way is to repeat the submatrices A_{ij} and A_{jj} reversing the signs of one of them to obtain the augmented matrix $\bar{A}$ as in the following

$$\bar{A} = \left[\begin{array}{cccccc|ccc} A_{1,1} & A_{1,2} & & & & A_{1,3} & A_{1,2} & A_{1,3} & \\ & A_{2,1} & A_{2,2} & A_{2,3} & & & -A_{2,1} & & A_{2,3} \\ & & & A_{3,2} & A_{3,3} & A_{3,1} & & -A_{3,1} & -A_{3,2} \end{array}\right]$$

Both ways make $\bar{A}_i \bar{A}_j{}^T$ zero for any pair i and j, and so the new matrix has mutually orthogonal partitions.

Running our solver in the augmented block Cimmino mode will go through the following steps:

- Partition the system into strips of rows (A_i and b_i for $i = 1 \ldots, p$)
- Augment the different partitions according to the selected algorithm
- Create the augmented systems
- Analyse and factorize the augmented systems using the direct solver MUMPS
- Build an auxiliary matrix S in parallel and use it to solve a reduced linear system. The result is then used to obtain the solution for the original linear system $Ax = b$.

For more details, we refer to [6, 9].

We consider the following row-blocks

$$\begin{bmatrix} A & C \\ B & S \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ f \end{bmatrix},$$

where x is ensured to be the same solution vector of $Ax = b$. We can denote by $\bar{A}$ the submatrix $[A\ C]$ where C as been chosen to enforce the p subspaces to be orthogonal as illustrated above, so that we have $\bar{A}^+ b = \sum_{i=1}^{p} A_i^+ b_j$. f and S are given by $f = -Y\bar{A}^+ b$ and $S = Y(I - P)Y^T$, with $Y = [0\ I]$. Finally the solution is given by

$$\begin{bmatrix} x \\ y \end{bmatrix} = \bar{A}^+ b + (I - P)Y^T S^{-1} f$$

because of mutual orthogonality between row-blocks $\bar{A}$ and $[B\ S]$.

To obtain the solution practically, we currently build S and factorize it using a direct solver. The added value of this approach is the fact that the columns of S can be built in an embarrassingly parallel fashion. The memory cost can be prohibitive in the case where S is not small or sparse enough, but we observe in many cases that S remains reasonable enough to make this approach computationally effective. The fact that S is symmetric positive definite also offers the possibility of computing $S^{-1} f$ iteratively using conjugate gradients, without building S explicitly.

5.2 Improvement achieved

Contributors	Daniel Ruiz (IRIT, WP1), Iain Duff (RAL-CERFACS, WP1), Philippe Leleux (CERFACS, WP1), Fahreddin Sukru Torun (IRIT-CNRS, WP1)

The package has been improved from software engineering, performance and maturity point of view. The following list summarizes our improvements:

- New load balancing algorithm for distributing row-blocks among MPI processes when there are more number of blocks than the number of MPI processes.
- Improved uniform partitioning method which works consistently for all kind of problems.
- Improved matrix scaling using parallel MC77 algorithm.

References

[1] Mario Arioli, Iain Duff, and Peter PM de Rijk. On the augmented system approach to sparse least-squares problems. *Numerische Mathematik*, 55(6):667–684, 1989.

[2] Mario Arioli, Iain Duff, Joseph Noailles, and Daniel Ruiz. A block projection method for sparse matrices. *SIAM Journal on Scientific and Statistical Computing*, 13(1):47–70, 1992.

[3] Mario Arioli, Iain S Duff, Daniel Ruiz, and Miloud Sadkane. Block lanczos techniques for accelerating the block cimmino method. *SIAM Journal on Scientific Computing*, 16(6):1478–1511, 1995.

[4] Randall Bramley and Ahmed Sameh. Row projection methods for large nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 13(1):168–193, 1992.

[5] LA Drummond, Iain S Duff, Ronan Guivarch, Daniel Ruiz, and Mohamed Zenadi. Partitioning strategies for the block cimmino algorithm. *Journal of Engineering Mathematics*, 93(1):21–39, 2015.

[6] Iain Duff, Ronan Guivarch, Daniel Ruiz, and Mohamed Zenadi. The augmented block cimmino distributed method. *SIAM Journal on Scientific Computing*, 37(3):A1248–A1269, 2015.

[7] Daniel Ruiz and Miloud Sadkane. Techniques for accelerating the block cimmino method. In *Proceedings of the Fifth SIAM Conference on Parallel Processing for Scientific Computing*, volume 62, page 98. SIAM, 1992.

[8] F. Sukru Torun, Murat Manguoglu, and Cevdet Aykanat. A novel partitioning method for accelerating the block cimmino algorithm. *CoRR*, abs/1710.07769, 2017.

[9] Mohamed Zenadi. *Méthodes hybrides pour la résolution de grands systèmes linéaires creux sur calculateurs parallèles.* PhD thesis, École Doctorale Mathématiques, Informatique et Télécommunications (Toulouse); 142547247, 2013.

6. AGMG

6.1 Package ID card

Package name	AGMG
Functionalities offered	Linear system solver
Description	AGMG implements an aggregation-based algebraic multigrid method. This method solves algebraic systems of linear equations, and is expected to be efficient for large systems arising from the discretization of scalar second order elliptic PDEs (see for [3, 1, 4, 2] for details and performance assessment). The method is however purely algebraic and may be tested on any problem. No information has to be supplied besides the system matrix and the right-hand-side.
Number of users	above 1000
Library dependencies	None
Package references	`http://homepages.ulb.ac.be/~ynotay/AGMG`
Contact	Yvan Notay (ynotay@ulb.ac.be)

6.2 Improvement achieved

Contributor	Yvan Notay (ULB)

A multithreaded version of the software package has been developed. Formerly (till release 3.2.4), AGMG was either sequential or MPI-based parallel. The latter version scales pretty well (see [5]), but requires that the matrix of the system to solve is distributed on as many MPI ranks as there are available cores. This is not suited when the program calling the AGMG solver is parallelized only via multithreading, or uses an hybrid MPI+OpenMP programming model.

The new multithreaded version (releases 3.3.0 and above) is either pure OpenMP or hybrid MPI+OpenMP. The calling sequence for the pure OpenMP variant is the same as that for the sequential version, whereas the calling sequence for the hybrid variant is the same as that for the pure MPI version. Thus, in particular, the pure OpenMP variant allows one to obtain parallel speedup from a purely sequential program.

The used parallelization strategy is the same as for the pure MPI version: unknowns and corresponding matrix rows are distributed among the threads, and most computations are kept inherently parallel by constraining the aggregation algorithm to aggregate only unknowns assigned to a same thread. The Gauss–Seidel smoothing procedure is also truncated to become inherently parallel.

The new multithreaded version has been assessed on the large test suite used as basis of development for AGMG. This latter is a collection of large sparse linear systems stemming from the discretization of second order elliptic PDEs, and comprising:

- Problems on 2D/3D regular grids and on 2D/3D unstructured grids, some of them with strong local refinement;
- Problems with (big) jumps and/or (large) anisotropy in the PDE coefficients;

- Symmetric (SPD) and nonsymmetric problems (2D/3D convection-diffusion with dominating convection);
- finite difference and finite element (up to $p4$) discretizations.

Matrix sizes range from 5×10^5 to $3. \times 10^7$, whereas the average number of nonzero entry per row ranges from 5. to 74. .

Timing results are displayed on Figure 3. One sees that for both sequential and multithreaded versions, the time per nonzero entry does not vary much despite the large variation in problems characteristics —the few pics correspond to challenging quasi singular convection-diffusion problems for which AGMG tends to outperform competitors, anyway. [1]

With the multithreaded version, the time needed per nonzero entry falls down to 0.1 microseconds on average. Using 8 cores, the speedup is roughly around 3.5 . This suboptimality is explained by the nature of the problem being solved: a sparse matrix problem with matrix stored in general sparse format and having only relatively few nonzero entries per row. It follows that the AGMG software code is strongly memory bound: beyond some point, having more computing power does not help if the memory bandwidth is not increased accordingly. (Observe that the test where ran on a simple workstation, without specific hardware enabling concurrent access of all cores to main memory.)

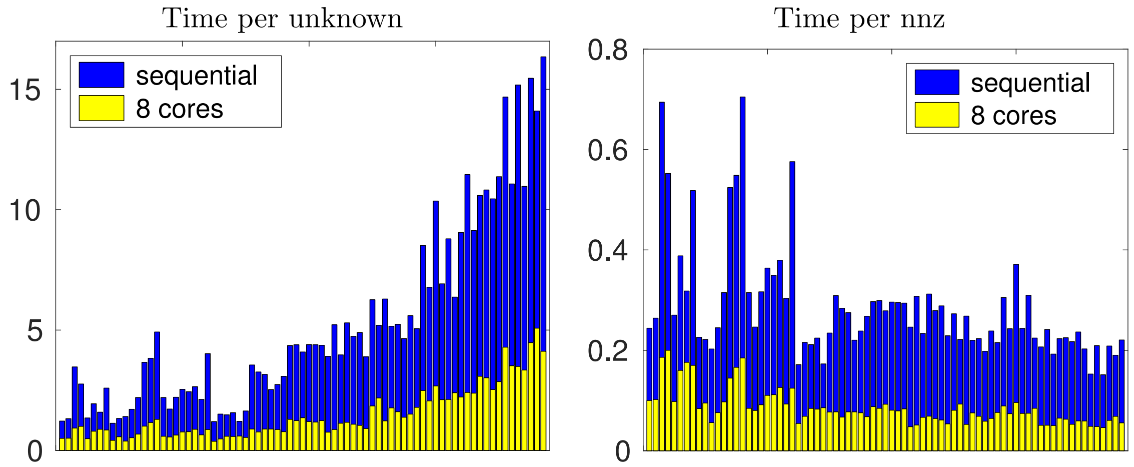

Figure 3: Total wall clock time to reduce the relative residual error by 10^{-6} – vs – problem index (problems ordered by increasing number of nonzero entry per row); times are reported in microseconds per unknown (left) or microseconds per nonzero entry (right); tests made on a desktop workstation – Intel XEON E5-2620 at 2.10GHz.

References

[1] A. Napov and Y. Notay, *An algebraic multigrid method with guaranteed convergence rate*, SIAM J. Sci. Comput., 34 (2012), pp. A1079–A1109.

[2] ——, *Algebraic multigrid for moderate order finite elements*, SIAM J. Sci. Comput., 36 (2014), p. A1678–A1707.

[1]see `http://homepages.ulb.ac.be/~ynotay/AGMG/perf.html`

[3] Y. Notay, *An aggregation-based algebraic multigrid method*, Electron. Trans. Numer. Anal., 37 (2010), pp. 123–146.

[4] ——, *Aggregation-based algebraic multigrid for convection-diffusion equations*, SIAM J. Sci. Comput., 34 (2012), pp. A2288–A2316.

[5] Y. Notay and A. Napov, *A massively parallel solver for discrete poisson-like problems*, J. Comput. Physics, 281 (2015), pp. 237–250.

7. PSBLAS and MLD2P4

7.1 Package ID card

Package name	PSBLAS
Functionalities offered	Parallel sparse linear algebra basic operators and iterative Krylov solvers
Description	PSBLAS (Parallel Sparse BLAS) is a library of Basic Linear Algebra Subroutines designed to handle the parallel implementation of iterative solvers for sparse linear systems. It includes functionalities for creating sparse matrices and handling their distribution and I/O, handling vectors associated with matrices, performing basic sparse matrix operations, and solving linear systems with a set of Krylov subspace methods. It is written in Fortran 2003, using MPI, and supports distributed sparse matrices in CSR, CSC, COO. Extensions for ELLPACK, JAD and GPU-enabled formats are also available. A plugin has been added to the library for efficient implementation of sparse matrix operations on GPUs.
Languages	Fortran 2003, interfaces to C and Octave in progress
Library dependencies	BLAS, MPI
Programing models	MPI, plugin for GPU available
Platforms	In the EoCoE project: • CRESCO cluster (ENEA) • IBM MareNostrum 4 (Barcelona Supercomputing Center) • Yoda Cluster (ICAR-CNR)
Code distribution	Available from `https://github.com/sfilippone/psblas3` under a modified BSD licence.
Package references	[1] S. Filippone, M. Colajanni, PSBLAS: A Library for Parallel Linear Algebra Computation on Sparse Matrices, ACM Trans. Math. Softw., 26, 2000, 527–550. [2] S. Filippone, A. Buttari, Object-Oriented Techniques for Sparse Matrix Computations in Fortran 2003, ACM Trans. on Math Software, 38, 2012, Art. No. 23. [3] V. Cardellini, S. Filippone, D. Rouson, Design Patterns for sparse-matrix computations on hybrid CPU/GPU platforms, Scientific Programming, 22, 2014, 1–19.
Contact	Salvatore Filippone (salvatore.filippone@cranfield.ac.uk)

Package name	MLD2P4
Functionalities offered	Parallel Algebraic MultiGrid and Domain Decomposition preconditioners
Description	MLD2P4 (MultiLevel Domain Decomposition Parallel Preconditioners Package based on PSBLAS) is a package of parallel Algebraic MultiGrid (AMG) and Domain Decomposition (multilevel additive and hybrid Schwarz) preconditioners. A decoupled version of the smoothed aggregation algorithm is applied to generate coarse-level corrections. MLD2P4 has been designed to provide scalable and easy-to-use preconditioners in the context of the PSBLAS computational framework and is used in conjuction with the PSBLAS Krylov solvers. MLD2P4 employs object-oriented design techniques in Fortran 2003, with interfaces to third party libraries such as MUMPS, UMFPACK, SuperLU, and SuperLU Dist, which can be exploited in building and applying AMG preconditioners.
Languages	Fortran 2003
Library dependencies	BLAS, MPI, PSBLAS, UMFPACK (optional), MUMPS (optional), SuperLU (optional), SuperLu_Dist (optional)
Programing models	MPI; GPU through PBLAS plugin
Platforms	In the EoCoE project: • CRESCO cluster (ENEA) • IBM MareNostrum 4 (Barcelona Supercomputing Center) • Yoda Cluster (ICAR-CNR)
Code distribution	Available from `https://github.com/sfilippone/mld2p4-2` under a modified BSD licence.
Package references	[1] P. D'Ambra, D. di Serafino, S. Filippone, MLD2P4: a Package of Parallel Algebraic Multilevel Domain Decomposition Preconditioners in Fortran 95, ACM Trans. Math. Softw., 37, 2010, Art. No. 30. [2] P. D'Ambra, D. di Serafino, S. Filippone, MLD2P4 v. 2.1 User's and Reference Guide, July 31, 2017. Available from `https://github.com/sfilippone/mld2p4-2/tree/development/docs`.
Contact	• Salvatore Filippone (salvatore.filippone@cranfield.ac.uk) • Pasqua D'Ambra (pasqua.dambra@cnr.it) • Daniela di Serafino (daniela.diserafino@unicampania.it)

7.2 Improvement achieved

Contributors	Pasqua D'Ambra (National Research Council of Italy - CNR, Naples, Italy), Daniela di Serafino (University of Campania "L. Vanvitelli", Caserta, Italy), Salvatore Filippone (Cranfield University, Cranfield, UK), Ambra Abdullahi Hassan (University of Rome "Tor Vergata", Rome, Italy

For each package, we first provide a short description of its status at the beginning of the EoCoE project and then outline the main improvements achieved. Results concerning the application of the current versions of MLD2P4 and PSBLAS to data sets from two different pillars of the EoCoE Project (Water for Energy - WP4, Meteorology for Energy - WP2) are described in Deliverable D1.3 (Application support outcome).

PSBLAS: starting point

The Parallel Sparse Basic Linear Algebra Subroutines (PSBLAS) library was designed to provide the operators needed to build iterative methods for the solution of sparse linear systems on distributed memory parallel computers. Its development was started taking into account the discussions on the standardization of sparse matrix computations in the context of the BLAS Technical Forum [9]. The library revolves around a set of Krylov subspace solvers for both symmetric positive definite (spd) and general matrices, e.g, Conjugate Gradients (CG), GMRES and BiCGSTAB, and a set of simple preconditioners including ILU(0).

The library contains a significant amount of infrastructure code to handle data storage and distribution of sparse matrices. Matrices are distributed in general row-block fashion, consistent with common usage of graph partitioning heuristics embodied in libraries such as Metis and SCOTCH; the data distribution can be specified in multiple ways. The necessary data exchange patterns and the global-to-local index remapping are automatically extracted from the matrix data: the halo data exchange, a typical step in mesh based computations, is provided as a communication primitive, and it is built to work for arbitrary distributed mesh graphs.

The parallel implementation is based on a Single Program Multiple Data (SPMD) paradigm and internally uses MPI, but provides wrappers for most common operations: user code rarely needs to invoke MPI directly. Similarly, the internal matrix storage formats are handled automatically by the library, including support for common formats such as CSR and COO, while at the same time providing tools to easily extend the set of supported formats. A set of plugins provides support for additional data storage formats such as ELLPACK and JAD, including storage formats that interface computations on NVIDIA GPUs [4, 11]. The design of the library is object-oriented, and implemented in Fortran 2003 [4, 10].

PSBLAS: improvement

The functionalities of PSBLAS have been extended during the EoCoE project, implementing a flexible version of the CG method (FCG) [12], and a variant of the Generalized Conjugate Residual method (GCR) [8, 13]. The former is equivalent to the standard Conjugate Gradient method when constant spd preconditioners are applied, and enhance the stability of the method when a variable preconditioner, such as the K-cycle available in MLD2P4 (see section 7.2), is employed. The latter applies to general linear systems and can be effectively used with variable preconditioning too.

We also included improvements needed when interfacing the GPU plugin [2, 4] with the MLD2P4 library described in the next section.

C and Octave interfaces to PSBLAS are under development and will be integrated in future versions of the library.

The curent version of PSBLAS (v. 3.5) is available from `https://github.com/sfilippone/psblas3`.

MLD2P4: starting point

MLD2P4 (MultiLevel Domain Decomposition Parallel Preconditioners Package based on PSBLAS was designed to provide scalable and easy-to-use algebraic multilevel domain decomposition preconditioners in the context of the PSBLAS (Parallel Sparse Basic Linear Algebra Subprograms) computational framework, for use with the Krylov solvers available from PSBLAS.

The release of MLD2P4 (MultiLevel Domain Decomposition Parallel Preconditioners Package based on PSBLAS) available at the beginning of the EoCoE project provided multilevel additive and hybrid Schwarz preconditioners, as well as one-level additive Schwarz preconditioners [5]. A purely algebraic approach, based on the *smoothed aggregation* algorithm [3, 16], was implemented to generate coarse-level corrections, so that no geometric background was needed about the matrix to be preconditioned. A decoupled version of this algorithm was considered, where the smoothed aggregation is applied locally to each submatrix [15].

The package employs object-oriented design techniques in Fortran 2003, with interfaces to additional third party libraries such as MUMPS, UMFPACK, SuperLU, and SuperLU_Dist, which can be exploited in building multi-level preconditioners. The parallel implementation is based on a SPMD paradigm; the inter-process data communication is based on MPI and is managed through PSBLAS primitives.

Several extensions and improvements have been introduced in MLD2P4 as a part of the EoCoE project.

MLD2P4: improvement

Several extensions and improvements have been introduced in MLD2P4 as a part of the EoCoE project, as specified next.

The package functionalities have been extended including multilevel cycles and smoothers widely used in multigrid methods. The classical V-cycle and W-cycle have been included in MLD2P4; furthermore, a K-cycle for both spd and general matrices has been implemented, where the coarse systems are solved by FCG(1) or GCR iterations at each level but the coarsest one [14, 13], in order to improve convergence when using unsmoothed constant piecewise prolongators. To enhance implementation scalability on linear systems coming from elliptic PDEs on regular grids, classical parallel pointwise smoothers have been added to the original additive Schwarz ones.

The user interface has been modified, in order to separate the construction of the multilevel hierarchy from the construction of the smoothers and solvers, and to allow for more flexibility at each level.

The software architecture has significantly evolved, in order to fully exploit the Fortran

2003 features implemented in PSBLAS 3.

Internal changes have been applied to MLD2P4 to guarantee optimal use of the GPU plugin available from PSBLAS.

MLD2P4 has been also interfaced with the compatible weighted matching aggregation algorithm implemented in the BootCMatch (Bootstrap AMG based on Compatible Weighted Matching) sequential code [7], obtaining a parallel decoupled version of this aggregation algorithm to be used within MLD2P4 for improving robustness and efficiency on sparse systems coming from anisotropic PDE problems on general grids [1]. Actually, this last issue is part of longer-term applied research work carried out within Task 2 of Workpackage 1. This work concerns the investigation of coarsening algorithms based on graph matching approaches in the AMG framework, and is motivated by the observation that the AMG preconditioners implemented in MLD2P4 may lose their robustness and parallel efficiency when applied to systems arising from highly anisotropic problems from EoCoE. A detailed description of this activity is provided in Deliverable D1.11.

The current stable version of MLD2P4 (v. 2.1) is available from `https://github.com/sfilippone/mld2p4-2`. (see [6] for a description of its functionalities). It includes all the previous improvements, but the interface with BootCMatch, which will be included in a future release.

References

[1] A. Abdullahi Hassan, P. D'Ambra, D. di Serafino, S. Filippone, *Parallel Aggregation Based on Compatible Weighted Matching for AMG*, in "Large-Scale Scientific Computing", I. Lirkov and S. Margenov eds., Lecture Notes in Computer Science, vol. 10665, Springer, 2018, pp. 563-571.

[2] D. Bertaccini and S. Filippone, *Sparse approximate inverse preconditioners on high performance GPU platforms*, Comput. Math. Appl., 71, 2016, 693–711.

[3] M. Brezina, P. Vaněk, *A Black-Box Iterative Solver Based on a Two-Level Schwarz Method*, Computing, 63, 1999, 233–263.

[4] V. Cardellini, S. Filippone, D. Rouson, *Design Patterns for sparse-matrix computations on hybrid CPU/GPU platforms*, Scientific Programming, 22, 2014, 1–19.

[5] P. D'Ambra, D. di Serafino, S. Filippone, *MLD2P4: a Package of Parallel Multilevel Algebraic Domain Decomposition Preconditioners in Fortran 95*, ACM Trans. Math. Softw., 37, 2010, Art. No. 30.

[6] P. D'Ambra, D. di Serafino, S. Filippone, *MLD2P4 v. 2.1 User's and Reference Guide*, July 31, 2017.

[7] P. D'Ambra, S. Filippone, P. S. Vassilevski, *BootCMatch: a Software Package for Bootstrap AMG based on Graph Weighted Matching*, ACM Trans. on Math Software, 44, 2018, Art. No. 39.

[8] S. C. Eisenstat, H. C. Elman, M. H. Schultz, *Variational iterative methods for non-symmetric systems of linear equations*, SIAM J. Numer. Anal., 20, 1983, 345-–357.

[9] S. Filippone, M. Colajanni, *PSBLAS: A Library for Parallel Linear Algebra Computation on Sparse Matrices*, ACM Trans. Math. Softw., 26, 2000, 527–550.

[10] S. Filippone, A. Buttari, *Object-Oriented Techniques for Sparse Matrix Computations in Fortran 2003*, ACM Trans. on Math Software, 38, 2012, Art. No. 23.

[11] S. Filippone, V. Cardellini, D. Barbieri, A. Fanfarillo, *Sparse Matrix-Vector Multiplication on GPGPUs*, ACM Trans. Math. Softw., 43, 2016, Art. No. 30.

[12] Y. Notay, *Flexible conjugate gradients*, SIAM J. Sci. Comput., 22, 2000, 1444–1460.

[13] Y. Notay, *An Aggregation-based Algebraic Multigrid Method*, Electron. Trans. Numer. Anal., 37, 2010, 123–146.

[14] Y. Notay, P. S. Vassilevski, *Recursive Krylov-based multigrid cycles*, Numer. Lin. Alg. Appl., 15, 2008, 473–487.

[15] R. S. Tuminaro, C. Tong, *Parallel Smoothed Aggregation Multigrid: Aggregation Strategies on Massively Parallel Machines*, in Proceedings of SuperComputing 2000 (J. Donnelley, Ed.), Dallas, 2000.

[16] P. Vaněk, J. Mandel and M. Brezina, *Algebraic Multigrid by Smoothed Aggregation for Second and Fourth Order Elliptic Problems*, Computing, 56, 1996, 179–196.

8. MUMPS

8.1 Package ID card

Package name	MUMPS
Functionalities offered	Parallel sparse direct solver
Description	MUMPS ("MUltifrontal Massively Parallel Solver") is a package for solving systems of linear equations of the form Ax = b, where A is a square sparse matrix that can be either unsymmetric, symmetric positive definite, or general symmetric, on distributed memory computers. It was developped inside a consortium started around CERFACS, INPT, inria, ENS-Lyon and Bordeaux-Univeristy.
Number of users	1-10
Library dependencies	MPI, BLAS, LAPACK, ScaLAPACK
Package references	http://mumps.enseeiht.fr/
Contact	• Fahreddin Sukru Torun (ftorun@enseeiht.fr) • Philippe Leleux (leleux@cerfacs.fr) • Mumps developers support (mumps-dev@listes.ens-lyon.fr)

8.2 Improvement achieved

MUMPS was used for Linear Algebra support of the applications Alya, ParFLOW, SHEMAT-Suite and TOKAM3X. In this section, we present an overview of the solver as well as its latest feature: Block Low Rank approximation which we used extensively for support.

Overview

MUMPS (MUltifrontal Massively Parallel direct Solver) is a package for solving systems of linear equations of the form Ax = b, where A is a sparse matrix. The solver has an Hybrid MPI/OpenMP model based on distributed dynamic scheduling, see [1] and [2] for more details.

MUMPS follows a multifrontal scheme, which is a direct method, composed of 3 steps:

- Analysis: preprocessing of the matrix (ordering, scaling, partitioning,...) and symbolic Factorisation. From the adjacency graph, this step allows the construction of an "elimination tree", decomposing the global system in smaller interconnected parts (fronts) for the factorisation. There exist 2 versions of this phase: one sequential and one parallel, we opted for the sequential option.
- Factorisation of the input matrix: this step makes use of 2 levels of parallelism, one introduced by the tree structure and the second is at node level where large fronts are solved by several processes.

- Solve: Forward/Backward substitution.

Block Low Rank Approximation

"Frontal matrices are not low-rank but in some applications they exhibit low-rank blocks. A block in the matrix represents the interaction between 2 subdomains. If they have a small diameter and are far away, their interaction is weak: the rank is low."
The goal is to approximate blocks far from the diagonal with low rank products so that we do not lose much information. This is done on blocks distant enough via a truncated Pivoted QR decomposition with a threshold (BLR epsilon), see [3] for more details.

When increasing Block Low Rank threshold parameter, more blocks are approximated and:

- Factorisation timing decreases with corresponding operations (Flops),
- Accuracy of the solution matches the threshold used (Scaled Residual).

MUMPS group has worked on exploiting BLR compression to also reduce the memory usage, Preliminary results are available in the Phd Thesis of Theo Mary[3], Section 9.3. This feature should be available early 2018 before a consortium release mid 2018.

References

[1] Patrick R Amestoy, Iain S Duff, Jean-Yves L'Excellent, and Jacko Koster. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001.

[2] Patrick R Amestoy, Abdou Guermouche, Jean-Yves L'Excellent, and Stéphane Pralet. Hybrid scheduling for the parallel solution of linear systems. *Parallel computing*, 32(2):136–156, 2006.

[3] Théo Mary. *Block Low-Rank multifrontal solvers: complexity, performance, and scalability.* PhD thesis, UT3, 2017.

9. Maphys

9.1 Package ID card

Package name	Maphys
Functionalities offered	Parallel sparse linear solveur
Description	Maphys is a hybrid direct/iterative solver that implements domain decomposition ideas at a pure algebraic form working only with the information associated with the user supplied sparse matrix.
Number of users	1-10 . . .
Library dependencies	MPI, MUMPS, PaStiX, BLAS, LAPACK, SCOTCH
Package references	`https://gitlab.inria.fr/solverstack/maphys/maphys`
Contact	• E. Agullo (emmanuel.agullo@inria.fr) • L. Giraud (luc.giraud@inria.fr) • M. Kuhn (matthieu.kuhn@inria.fr) • G. Marait (gilles.marait@inria.fr) • L. Poirel (louis.poirel@inria.fr)

In this section we describe the design of the hybrid solver `MaPHyS` a non-overlapping domain decomposition. For the sake of simplicity, we assume that $\mathcal{A}$ has a symmetric pattern. The `MaPHyS` package is available on the following git server:

https://gitlab.inria.fr/solverstack/maphys/maphys.

Let $\mathcal{A}x = b$ be the linear problem and $\mathcal{G} = \{V, E\}$ the adjacency graph associated with $\mathcal{A}$. In this graph, each vertex is associated with a row or column of the matrix $\mathcal{A}$ and it exists an edge between the vertices i and j if the entry $a_{i,j}$ is non zero. In the sequel, to facilitate the exposure and limit the notation we voluntarily mix a vertex of $\mathcal{G}$ with its index depending on the context of the description. The governing idea behind substructuring or Schur complement methods is to split the unknowns in two categories: interior and interface vertices. We assume that the vertices of the graph $\mathcal{G}$ are partitioned into $\mathcal{N}$ disconnected subgraphs $\mathcal{I}_1$, ..., $\mathcal{I}_\mathcal{N}$ separated by the global vertex separator Γ. We also decompose the vertex separator Γ into non-disjoint subsets Γ_i, where Γ_i is the set of vertices in Γ that are connected to at least one vertex of $\mathcal{I}_i$. Notice that this decomposition is not a partition as $\Gamma_i \cap \Gamma_j \neq \emptyset$ when the set of vertices in this intersection defines the separator of $\mathcal{I}_i$ and $\mathcal{I}_j$. By analogy with classical domain decomposition in a finite element framework, $\Omega_i = \mathcal{I}_i \cup \Gamma_i$ will be referred to as a subdomain with internal unknowns $\mathcal{I}_i$ and interface unknowns Γ_i. If we denote $\mathcal{I} = \cup \mathcal{I}_i$ and order vertices in $\mathcal{I}$ first, we obtain the following block reordered linear system

$$\begin{pmatrix} \mathcal{A}_{\mathcal{I}\mathcal{I}} & \mathcal{A}_{\mathcal{I}\Gamma} \\ \mathcal{A}_{\Gamma\mathcal{I}} & \mathcal{A}_{\Gamma\Gamma} \end{pmatrix} \begin{pmatrix} x_\mathcal{I} \\ x_\Gamma \end{pmatrix} = \begin{pmatrix} b_\mathcal{I} \\ b_\Gamma \end{pmatrix} \tag{7}$$

where x_Γ contains all unknowns associated with the separator and $x_\mathcal{I}$ contains the unknowns associated with the interiors. Because the interior vertices are only connected to either interior vertices in the same subgraph or with vertices in the interface, the matrix $\mathcal{A}_{\mathcal{I}\mathcal{I}}$ has a block diagonal structure, where each diagonal block corresponds to one subgraph $\mathcal{I}_i$. Eliminating $x_\mathcal{I}$ from the second block row of Equation (7) leads to the reduced system

$$\mathcal{S}x_\Gamma = f \tag{8}$$

where

$$\mathcal{S} = \mathcal{A}_{\Gamma\Gamma} - \mathcal{A}_{\Gamma\mathcal{I}}\mathcal{A}_{\mathcal{I}\mathcal{I}}^{-1}\mathcal{A}_{\mathcal{I}\Gamma} \text{ and } f = b_\Gamma - \mathcal{A}_{\Gamma\mathcal{I}}\mathcal{A}_{\mathcal{I}\mathcal{I}}^{-1}b_{\mathcal{I}}. \tag{9}$$

The matrix $\mathcal{S}$ is referred to as the *Schur complement matrix.* This reformulation leads to a general strategy for solving (7). Specifically, an iterative Krylov subspace method can be applied to solve (8) using a stopping criterion

$$\frac{\|\mathcal{S}x_\Gamma - f\|}{\|b} \approx \frac{\|\mathcal{A}x - b\|}{\|b\|} \leq \varepsilon.$$

Once x_Γ is known, $x_{\mathcal{I}}$ can be computed with one additional solve for the interior unknowns via

$$x_{\mathcal{I}} = \mathcal{A}_{\mathcal{I}\mathcal{I}}^{-1}\left(b_{\mathcal{I}} - \mathcal{A}_{\mathcal{I}\Gamma}x_\Gamma\right).$$

The local interiors are disjoint and form a partition of the interior $\mathcal{I} = \sqcup\mathcal{I}_i$. It is not necessarily the case for the boundaries. Indeed, two subdomains Ω_i and Ω_j may share part of their interface ($\Gamma_i \bigcap \Gamma_j \neq \emptyset$). Altogether, the local boundaries form the overall interface $\Gamma = \cup\Gamma_i$ which is not a disjoint union. Because interior vertices are only connected to vertices of their subset (either on the interior or on the boundary), matrix $\mathcal{A}_{\mathcal{I}\mathcal{I}}$ associated to the interior has a block diagonal structure. Each diagonal block $\mathcal{A}_{\mathcal{I}_i\mathcal{I}_i}$ corresponds to a local interior.

While the Schur complement system is significantly smaller and better conditioned than the original matrix $\mathcal{A}$, it is important to consider further preconditioning when employing a Krylov method. We introduce the general form of the preconditioner considered in `MaPHyS`. To describe the main preconditioner in `MaPHyS`, we define $\bar{\mathcal{S}}_i = \mathcal{R}_{\Gamma_i}\mathcal{S}\mathcal{R}_{\Gamma_i}^T$, that corresponds to the restriction of the Schur complement to the interface Γ_i. If $\mathcal{I}_i$ is a fully connected subgraph of $\mathcal{G}$, the matrix $\bar{\mathcal{S}}_i$ is dense.

With these notations the Additive Schwarz preconditioner reads

$$\mathcal{M}_{AS} = \sum_{i=1}^{\mathcal{N}} \mathcal{R}_{\Gamma_i}^T \bar{\mathcal{S}}_i^{-1} \mathcal{R}_{\Gamma_i}. \tag{10}$$

We notice that this preconditioner has a form similar to the Neumann-Neumann preconditioner [3, 6], but in the SPD case $\mathcal{M}_{AS}$ is always defined and SPD (as $\mathcal{S}$ is SPD [5]); which is not always the case for Neumann-Neumann.

If we considered a planar graph partitioned into horizontal strips (1D decomposition), the resulting Schur complement matrix has a block tridiagonal structure as depicted in (11)

$$\mathcal{S} = \begin{pmatrix} \ddots & & & & \\ & \mathcal{S}_{k,k} & \mathcal{S}_{k,k+1} & & \\ & \mathcal{S}_{k+1,k} & \mathcal{S}_{k+1,k+1} & \mathcal{S}_{k+1,k+2} & \\ & & \mathcal{S}_{k+1,k+2} & \mathcal{S}_{k+2,k+2} & \\ & & & & \ddots \end{pmatrix}. \tag{11}$$

For that particular structure of $\mathcal{S}$, the submatrices in boxes correspond to the $\bar{\mathcal{S}}_i$ local restriction of the Schur $\mathcal{S}$. Such diagonal blocks, which overlap with one another, are similar to the classical block overlap of the Schwarz method when writing in a matrix form for 1D decomposition. Similar ideas have been developed in a pure algebraic context in earlier papers (e.g., [4]) for the solution of general sparse linear systems. Because of this link, the preconditioner defined by (10) is referred to as algebraic additive Schwarz for the Schur complement.

`MaPHyS` is based on an algebraic domain decomposition whose primary motivation is to naturally exploit some parallelism between the computation performed on each sub-problem of the decomposition using MPI. It can also exploit a second level of parallelism at the subdomain level using threads.

Based on the decomposition of $\mathcal{G}$ we can define a decomposition of the matrix $\mathcal{A}$ where each sub-matrix is associated with a subdomain and is allocated to one MPI process. Notice that due to the overlap between local interfaces Γ_i, a special attention has to be paid to the decomposition of $\mathcal{A}_{\Gamma\Gamma}$ as its entries are shared between different processes. In that respect the matrix entries of $\mathcal{A}_{\Gamma\Gamma}$ must be weighted so that the sum of the coefficients on the local interface submatrices are equal to one. For that, we introduce the *weighted local interface* matrix $\mathcal{A}^w_{\Gamma_i\Gamma_i}$ that satisfies $\mathcal{A}_{\Gamma\Gamma} = \sum_{i=1}^{\mathcal{N}} \mathcal{R}^T_{\Gamma_i}\mathcal{A}^w_{\Gamma_i\Gamma_i}\mathcal{R}_{\Gamma_i}$, where $\mathcal{R}_{\Gamma_i} : \Gamma \to \Gamma_i$ is again the canonical point-wise restriction which maps full vectors defined on Γ into vectors defined on Γ_i. In matrix terms, a subdomain Ω_i may then be represented by the *local matrix* $\mathcal{A}_i$ defined by

$$\mathcal{A}_i = \begin{pmatrix} \mathcal{A}_{\mathcal{I}_i\mathcal{I}_i} & \mathcal{A}_{\mathcal{I}_i\Gamma_i} \\ \mathcal{A}_{\Gamma_i\mathcal{I}_i} & \mathcal{A}^w_{\Gamma_i\Gamma_i} \end{pmatrix}. \tag{12}$$

The global Schur complement matrix $\mathcal{S}$ from (8) can then be written as the sum of elementary matrices

$$\mathcal{S} = \sum_{i=1}^{\mathcal{N}} \mathcal{R}^T_{\Gamma_i}\mathcal{S}_i\mathcal{R}_{\Gamma_i} \tag{13}$$

where

$$\mathcal{S}_i = \mathcal{A}^w_{\Gamma_i\Gamma_i} - \mathcal{A}_{\Gamma_i\mathcal{I}_i}\mathcal{A}^{-1}_{\mathcal{I}_i\mathcal{I}_i}\mathcal{A}_{\mathcal{I}_i\Gamma_i} \tag{14}$$

is the *local Schur complement* associated to subdomain Ω_i. This local expression allows for computing local Schur complements independently from each other.

The $\bar{\mathcal{S}}_i$'s that are involved in the definition of $\mathcal{M}_{AS}$ can actually be built within this data distribution from the $\mathcal{S}_i$'s. Let us simply describe this calculation on a simple example for a given subdomain Ω_i with four neighbors $\Omega_m, \Omega_g, \Omega_k$ ans Ω_ℓ. The local Schur complement matrix associated with Ω_i is dense and has the following 4×4 block structure

$$\mathcal{S}_i = \begin{pmatrix} \mathcal{S}^{(i)}_{mm} & \mathcal{S}_{mg} & \mathcal{S}_{mk} & \mathcal{S}_{m\ell} \\ \mathcal{S}_{gm} & \mathcal{S}^{(i)}_{gg} & \mathcal{S}_{gk} & \mathcal{S}_{g\ell} \\ \mathcal{S}_{km} & \mathcal{S}_{kg} & \mathcal{S}^{(i)}_{kk} & \mathcal{S}_{k\ell} \\ \mathcal{S}_{\ell m} & \mathcal{S}_{\ell g} & \mathcal{S}_{\ell k} & \mathcal{S}^{(i)}_{\ell\ell} \end{pmatrix} \tag{15}$$

where each block accounts for the interactions between the unknowns on the edges of its interface. The matrix $\bar{\mathcal{S}}_i$ can be built from the local Schur complement $\mathcal{S}_i$ by assembling its diagonal blocks thanks to a few neighbour to neighbour communications. For instance, the diagonal blocks of $\mathcal{S}_i$ associated with the interface E_k between Ω_i and Ω_j is $\mathcal{S}_{kk} = \mathcal{S}^{(i)}_{kk} + \mathcal{S}^{(j)}_{kk}$. Assembling each diagonal block of the local Schur complement matrices, we obtain the local assembled Schur complement, that is

$$\bar{\mathcal{S}} = \begin{pmatrix} \mathcal{S}_{mm} & \mathcal{S}_{mg} & \mathcal{S}_{mk} & \mathcal{S}_{m\ell} \\ \mathcal{S}_{gm} & \mathcal{S}_{gg} & \mathcal{S}_{gk} & \mathcal{S}_{g\ell} \\ \mathcal{S}_{km} & \mathcal{S}_{kg} & \mathcal{S}_{kk} & \mathcal{S}_{k\ell} \\ \mathcal{S}_{\ell m} & \mathcal{S}_{\ell g} & \mathcal{S}_{\ell k} & \mathcal{S}_{\ell\ell} \end{pmatrix}.$$

The original idea of hybrid methods based on DDM consists into subdividing the graph into subgraphs that are individually mapped to one process. This approach is referred to as the classical parallel implementation.

With all these components, the classical parallel implementation of `MaPHyS` can be decomposed into four main phases:

- the partitioning step consists of partitioning the adjacency graph $\mathcal{G}$ of $\mathcal{A}$ into several subdomains and distribute the $\mathcal{A}_i$ to different cores;
- the factorization of the interiors and the computation of the local Schur complement $\mathcal{S}_i$ using $\mathcal{A}_i$;
- the setup of the preconditioner by assembling diagonal blocks of $\mathcal{S}_i$ via a few neighbour to neighbour communication and factorization of this one. It is also during this step that the coarse grid correction is computed when the corresponding option is set;
- the solve step where a parallel preconditioned Krylov method is performed on the reduced system (Equation 8) to compute x_{Γ_i} followed by the back solve on the interior to compute $x_{\mathcal{I}_i}$.

9.2 Improvement achieved

Contributors	E. Agullo (Inria), L. Giraud (Inria), M. Kuhn (Inria), G. Marait (Inria), L. Poirel (Inria).

We made a few progresses from version 0.9.4.2, on various components of the software package addressing differents aspects:

1. New software deployment service on top of Spack to automatise the installation of the package and its numerous dependencies.
2. Replace the dedicated matrix partition by a more modular and flexible parallel partitioning/data distribution module.
3. Integrate a prototype of the new algebraic coarse space for SPD matrices to control the condition number [1]. A description of this feature is available below.
4. Design a new API to interface Maphys with newly developped block Krylov solvers for multiple right-hand sides [2].
5. Option to keep the same preconditionner when MaPHyS driver is called several time on different matrices. This is especially useful to solve non-linear simulation cases when the matrix changes little between iterations (this feature has been tested in AlyA).

These improvements have been tested and integrated in version 0.9.7 of MaPHyS.

Maphys coarse grid correction mechanism. The coarse grid correction of Maphys consists of building a relevant coarse space for the iterative solution part. This coarse space is built by solving independent generalized eigenproblem (one per subdomain), whose goal is to control the condition number of the preconditioned Schur system. Once the coarse space is built, a coarse preconditioner is computed. Then, the coarse preconditioner is applied in an additive way to the classical preconditioned CG algorithm of Maphys, resulting in a so called 2-level additive Schwarz preconditioner. For more details on the coarse grid correction of Maphys, please refer to Poirel et al., 2016.

In practice, when using the coarse grid correction into Maphys, the quality of the condition number of the Schur system is controlled by setting a desired number of eigenvalues/eigenvectors pairs to be computed for each subdomain for the coarse space computation. To this end, there are four available implementation strategies to apply the coarse grid correction through the coarse preconditioner application, presented in the following list by chronological order of implementation:

- The first one consists in using Mumps sparse direct solver with its distributed input mode. Into this mode, the preconditioner is built and applied using all the available MPI processes reachable into Maphys communicator.
- The second one consists in using dense direct solver. Here, the coarse problem is solved in a centralized way, on one MPI process.
- The third one consists in using a sparse direct solver with a centralized input. The coarse problem can then solved on a sub-communicator of Maphys MPI communicator with any desired number of ressources.
- The last one extends the previous implementation strategy by allowing to duplicate the coarse problem on disjoint and equally sized sub-communicators of Maphys MPI communicator. This strategy allows to bypass a global MPI communication after the coarse preconditioner application while keeping control on the memory overhead due to the coarse problem duplication.

The coarse grid correction feature has been successfully tested up to one billion unknowns (on 667 nodes; 16,000 cores) in weak scaling on CINES's OCCIGEN cluster.

References

[1] Emmanuel Agullo, Luc Giraud, and Yan-Fei Jing. Block GMRES method with inexact breakdowns and deflated restarting. *SIAM Journal on Matrix Analysis and Applications*, 35(4):1625–1651, November 2014.

[2] Emmanuel Agullo, Luc Giraud, and L Poirel. Robust coarse spaces for Abstract Schwarz preconditioners via generalized eigenproblems. Research Report RR-8978, INRIA Bordeaux, November 2016.

[3] J.-F. Bourgat, R. Glowinski, P. Le Tallec, and M. Vidrascu. Variational formulation and algorithm for trace operator in domain decomposition calculations. In Tony Chan, Roland Glowinski, Jacques Périaux, and Olof Widlund, editors, *Domain Decomposition Methods*, pages 3–16, Philadelphia, PA, 1989. SIAM.

[4] X.-C. Cai and Y. Saad. Overlapping domain decomposition algorithms for general sparse matrices. *Numerical Linear Algebra with Applications*, 3:221–237, 1996.

[5] L. M. Carvalho, L. Giraud, and G. Meurant. Local preconditioners for two-level non-overlapping domain decomposition methods. *Numerical Linear Algebra with Applications*, 8(4):207–227, 2001.

[6] Y.-H. De Roeck and P. Le Tallec. Analysis and test of a local domain decomposition preconditioner. In Roland Glowinski, Yuri Kuznetsov, Gérard Meurant, Jacques Périaux, and Olof Widlund, editors, *Fourth International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 112–128. SIAM, Philadelphia, PA, 1991.